Warming-up recurrent neural networks to maximize reachable multi-stability greatly improves learning

Nicolas Vecoven  
Department of Montefiore  
University of Liège  
nvecoven@uliege.be

Damien Ernst  
Department of Montefiore  
University of Liège  
dernst@uliege.be

Guillaume Drion  
Department of Montefiore  
University of Liège  
gdrion@uliege.be

Abstract

Training recurrent neural networks is known to be difficult when time dependencies become long. Consequently, training standard gated cells such as gated recurrent units and long-short term memory on benchmarks where long-term memory is required remains an arduous task. In this work, we propose a general way to initialize any recurrent network connectivity through a process called “warm-up” to improve its capability to learn arbitrarily long time dependencies. This initialization process is designed to maximize network reachable multi-stability, i.e. the number of attractors within the network that can be reached through relevant input trajectories. Warming-up is performed before training, using stochastic gradient descent on a specifically designed loss. We show that warming-up greatly improves recurrent neural network performance on long-term memory benchmarks for multiple recurrent cell types, but can sometimes impede precision. We therefore introduce a parallel recurrent network structure with partial warm-up that is shown to greatly improve learning on long time-series while maintaining high levels of precision. This approach provides a general framework for improving learning abilities of any recurrent cell type when long-term memory is required.

1 Introduction

Despite their performances and widespread use, recurrent neural networks (RNNs) have been known to be blackbox models with extremely complex internal dynamics. A growing body of works has focused on understanding the internal dynamics of trained RNNs (Sussillo and Barak [2013], Ceni et al. [2020], Maheswaranathan et al. [2019]), providing invaluable insights into the RNN prediction process. In particular, recent work has highlighted the important role played by fixed points in RNNs, and has argued that efficiently locating such fixed points could provide insights on RNN dynamics and input-output properties (Katz and Reggia [2017]). Here, we build upon this line of work to highlight how maximizing the number of reachable fixed points before learning improves RNN learning, in particular in the presence of arbitrarily long time dependencies.

Training RNNs has been shown to be difficult when time dependencies become too long (Pascanu et al. [2013]). This limitation is often due to vanishing or exploding gradients. Different methods have been proposed to help this issue, such as introducing gating mechanisms (Cho et al. [2014], Hochreiter and Schmidhuber [1997]), clipping gradients (Pascanu et al. [2013]) and also maintaining orthogonality in recurrent weight matrices (Jing et al. [2019]). Other authors also proposed new
ways of initialising RNN parameters (Pascanu et al., 2013; Van Der Westhuizen and Lasenby, 2018; Marichal et al., 2009), endowing them with better training properties. Likewise, novel recurrent cell dynamics, such as neuromodulated bistable recurrent cells (nBRC, Vecoven et al., 2020), have been introduced to help tackle long-term memory benchmarks. nBRC were specifically designed to maximize reachability of cellular bistability, providing a way to create never-fading memory at the cellular level. Promising results highlighted how dynamics of untrained RNNs, i.e. at initialization, can strongly impact RNNs learning performance during training.

In this work, we extend this approach at the network level by maximizing reachable multi-stability of any recurrent cell type prior to learning. To this end, we propose a novel RNN pre-training procedure called “warm-up” that is designed to maximize the number of RNN attractors that can be reached by a set of training samples. First, we introduce a fast-to-compute and differentiable measure called variability amongst attractors (VAA) that counts the number of different reachable attractors within a network. We show that loss decrease during learning in long-term memory benchmarks is highly correlated with an increase in VAA, highlighting both the relevance of the measure and the importance of multi-stability for efficient learning. Second, we use stochastic gradient ascent on VAA before training as a way to maximize the number of reachable attractors within the network. We show that this technique strongly improves performance on long-term memory benchmarks, though at the cost of precision, the latter relying on the richness of network transient dynamics. Finally, we propose a parallel recurrent network structure with partial warm-up that enables the combining of long-term memory through multi-stability on the one hand, and precision through rich transient dynamics on the other hand. For multiple RNNs such as gated recurrent units (GRUs (Cho et al., 2014)) and long-short term memory (LSTMs (Hochreiter and Schmidhuber, 1997)), we show that this method indeed retains the benefits of warm-up, while increasing predictive performances.

2 Background

2.1 Recurrent neural networks

RNNs are parametric function approximators that are often used to tackle problems with temporal structure. Their architecture is similar to that of standard multilayer perceptrons with added temporal connections, that is, each layer is connected to itself through time. This connection allows RNNs to memorize relevant information that can only be captured over multiple time-steps. More formally, let \( U = \{u_0, \ldots, u_T\} \) with \( T \in \mathbb{N}_0 \) and \( u_t \in \mathbb{R}^n \) denote a time-series. RNNs maintain an internal memory state \( x_t \) through an update rule \( x_t = f(x_{t-1}, u_t; \theta) \) and output a value \( o_t = g(x_{t-1}, u_t; \theta) \), where \( x_0 \) is a constant and \( \theta \) are the parameters of the network. We note that frequently, \( x_t = o_t \). Such networks are usually built with multiple layers that are linked sequentially through \( u_t^i = o_t^{i-1} \) with \( u_0^i = u_i \), where \( o_t^i \) denotes the output of layer \( i \) and \( u_t^i \) its input. Backpropagation through time (BPTT) is used to train these networks where gradients are computed through the unrolled model. In practice, it is difficult to do this on longer time-series, due to vanishing and exploding gradients. For an in-depth study on the subject and some proposed solutions, we refer the reader to (Pascanu et al., 2013).

2.2 Fixed points and attractors in RNNs

Due to their temporal nature and update rules, RNNs can be seen as dynamical systems. This viewpoint has already been used to understand the difficulties for RNNs to capture longer time dependencies (Bengio et al., 1993; Doya, 1993). Recently, it was shown that fixed points are important when it comes to understanding the prediction process of trained RNNs (Sussillo and Barak, 2013). Fixed points are well-known characteristics of dynamical systems that are defined as points in the phase space that map to themselves through the update function. Fixed points depend on the input of the system. In the case of RNNs, we say that a state \( x^* \) is a fixed point of a network in \( u \) if and only if:

\[
x^* = f(x^*, u; \theta)
\]

Fixed points can either be fully attractive (attractors), fully repulsive (repellors), or combine attractive and repulsive manifolds (saddle points). Attractors and saddle points are the most useful when it comes to understanding RNNs dynamics. Attractors correspond to network steady-states and are
thought to be the allowing factor for RNNs to maintain long-term information (Pascanu et al., 2013; Sussillo and Barak, 2013; Maheswaranathan et al., 2019). They are defined as fixed points towards which the system converges for multiple starting conditions. The set of starting states for which the system converges to the attractor \( x^* \) is called basin of attraction of \( x^* \) and written as \( B_{x^*} \). Basins of attraction are delimited by the stable manifolds of saddle-points. Mathematically, \( x^* \) is an attractor in \( u \) if its basin of attraction in \( u \), \( B_{x^*}^u \), is not a singleton and is such that:

\[
    x \in B_{x^*}^u \iff \lim_{n \to \infty} f^n(x, u; \theta) = x^* .
\]

From this definition, one can introduce the notion of reachable attractors. In particular, we say that an attractor \( x^* \) is reachable in \( u \) if there exists an input to the system such that its final state belongs to \( B_{x^*}^u \). More formally, an attractor \( x^* \) is said to be reachable in \( u \) if there exists \( U^* = [u^*_1, \ldots, u^*_M] \) such that \( x_M \in B_{x^*}^u \), where \( x_{i+1} = f(x_i, u_i) \) with \( x_1 = u_1 \) and \( M \in \mathbb{N} \). A system that has a single reachable attractor in \( u \) is said to be monostable in \( u \), whereas a system that has multiple reachable attractors in \( u \) is said to be multi-stable in \( u \).

### 3 Variability amongst attractors and warm-up

We propose to introduce VAA as a proxy measure for counting the number of attractors in dynamical systems. We first detail how VAA can be computed for any arbitrary dynamical system and proceed by detailing how we use VAA in the case of RNNs. We then show how an increase in VAA is highly correlated to loss decrease during RNN training on a denoising benchmark. Finally, we detail the warm-up procedure, explaining how VAA can be used to maximize the number of reachable attractors in RNNs.

#### Variability amongst attractors.

VAA is quite straightforward in essence. Given a batch of different initial-state conditions and a constant perturbation, VAA is computed as the proportion of different states towards which the system converges. States are considered different if their Euclidian distance in phase space is greater than a given threshold \( \epsilon \in \mathbb{R} \). Concretely, we sample a perturbation \( u \) and \( n \in \mathbb{N}_0 \) initial states \( \{x_{0,1}; \ldots; x_{0,n}\} \) in phase space where \( n \) is a hyper-parameter. We detail later how we define these distributions for RNNs. For each initial state, we let the model converge over \( M \) time-steps with \( u \) as input. If we define \( \delta \) as the minimal Euclidian distance between two attractors, \( M \) must be chosen large enough such that the Euclidian distance between all \( x_{M,i} \) and their corresponding attractor is smaller than \( \delta/4 \). This ensures that there exists a threshold \( \epsilon \) that captures all states belonging to a same attractor while ensuring zero overlap between states belonging to different attractors. Finally, we count the number of unique vectors (given \( \epsilon \)) in the final states \( \{x_{M,1}; \ldots; x_{M,n}\} \). This is done by first building a correspondence matrix \( C \) where \( C_{i,j} \) is equal to 1 if \( x_{M,i} \) is close enough to \( x_{M,j} \) in Euclidian distance and else to 0. From this matrix, we build a vector \( v \) such that \( v[i] \) is equal to the number of corresponding states to \( x_{M,i} \). We can observe that \( \forall i \in \{1, \ldots, n\} \) elements are equal to \( v[i] \). Indeed, let \( m \) be the number of reached final states associated to an attractor. By definition each of these \( m \) states will be similar to \( m - 1 \) other states. Thus, the number of unique states can simply be computed by inverting each element in \( v \) and summing them. Once the number of different states has been computed, we divide it by \( n \) to obtain the proportion of different states, and thus the VAA. Alternatively put:

\[
    \begin{align*}
        &\left\{ \begin{array}{l}
            C_{i,j} = 1 \text{ if } ||x_{M,i} - x_{M,j}|| < \epsilon \\
            \forall i, j \in \{1, \ldots, n\}^2
        \end{array} \right.
        \forall i \in \{1, \ldots, n\}
        \sum_{j=1}^{n} C_{i,j}
    \\
        &v[i] = \sum_{j=1}^{n} C_{i,j}
    \\
        &VAA = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{v[i]}
    \end{align*}
\]

We note that VAA will vary between \( \frac{1}{n} \) and 1. Indeed, if the system is mono-stable (has only one attractor) then the VAA will be equal to \( \frac{1}{n} \), whereas if the network converges to different states for all input trajectories, VAA will be equal to 1.

When measuring the VAA on RNNs, we are interested in the number of attractors that can be reached when receiving time-series from a given dataset as inputs. For this reason, when computing the VAA, we use a batch of \( n \) samples from the dataset used for training the model, where each sample corresponds to a time-series \( U_i = \{u_{0,i}, \ldots, u_{T_i}\} \). Each time-series is then truncated at random before being fed to the RNN. The resulting final states of the system (one per time-series) are used
as initial states for the VAA procedure detailed above. Furthermore, the perturbation $u$ is sampled from a multivariate normal distribution. By repeating the VAA procedure multiple times and if VAA is greater than $1/n$ for all different sampled perturbations, we can then assume that the RNN is multi-stable for a wide range of perturbations. The full procedure for computing the VAA of an RNN is presented in Algorithm 1.

![Algorithm 1: Computing VAA for an RNN](image)

In terms of computations, measuring VAA is similar to a batched forward pass of the network and is thus very efficient, allowing its computation during training. To illustrate the relevance of this measure, we train a GRU network on a denoising benchmark (see Section 4 for full details on the benchmark). Figure 1 shows a strong correlation between the ability of the network to learn (thus, a decrease in the mean-squared error) and its measured VAA. Interestingly, we note that each VAA measure reported in the Figure is computed with different perturbations, which suggests that when RNN become multi-stable, it is for a wide range of perturbations. Furthermore, we can also observe that GRU are mono-stable at initialization and that multiple gradient steps are required to reach multi-stability. These observations lead us to propose warm-up, a procedure in which VAA is used to promote multi-stability in RNNs before training.

**Warming-up.** The goal of warm-up is to maximize the number of reachable attractors of a RNN for a given dataset, that is, to maximize VAA. As is usually done for training neural networks, we...
propose using stochastic gradient descent (SGD) so as to maximize \( VAA \). We note, however, that SGD cannot be used directly on the \( VAA \) measure detailed in Algorithm 1 for two different reasons.

- First, we note that the deeper recurrent neural networks are, the bigger \( M \) must become for reaching convergence. Indeed, one must wait for the shallower layers to converge before reaching convergence of deeper layers. This is still practical for computing \( VAA \) but can become too expensive for propagating the gradient back through time on such long sequences.
- Second, we note that the way the correspondence matrix \( C \) is built is not differentiable.

To solve the first problem, when warming up, we treat each layer as a separate dynamical system and we maximize the number of reachable attractors for each layer independently.

Solving the second problem can easily be done by introducing a differentiable proxy measure for \( VAA \). We denote the differentiable proxy measure for computing \( VAA \) as \( VAA^* \). The only difference with the \( VAA \) measure is in the definition of \( C \) as that is the only non-differentiable computation. For all \( i, j \) in \( \{1, \ldots, n\} \), we approximate \( C \) as:

\[
\begin{align*}
\hat{C}_{i,j} &= 1 - \frac{\max(0, ||\tanh(x_{M,i}) - \tanh(x_{M,j})||) - \varepsilon}{||\tanh(x_{M,i}) - \tanh(x_{M,j})||} \quad \text{if } i \neq j \\
\hat{C}_{i,j} &= 1 \text{ else.}
\end{align*}
\]

We note that the value of \( \hat{C}_{i,j} \) is strictly equal to 1 if \( x_{M,i} \) is close enough in Euclidian distance to \( x_{M,j} \). On the other hand, \( \hat{C}_{i,j} \) will be close to 0 when \( x_{M,i} \) and \( x_{M,j} \) are different. We note that the \( \hat{C}_{i,j} \) will never strictly be equal to 0, but will get closer as the distance between \( x_{M,i} \) and \( x_{M,j} \) increases since \( \frac{||\tanh(x_{M,i}) - \tanh(x_{M,j})|| - \varepsilon}{||\tanh(x_{M,i}) - \tanh(x_{M,j})||} \) will get closer to 1. Although we are not directly interested in states being far apart from each other, we noticed that this small bias in favor of \( VAA^* \) increasing slightly as distance between states grows provides a good direction to the gradient for reaching multi-stability. Furthermore, we note that this bias encourages using a saturating function (a hyperbolic tangent in this case) on the states. It permits to saturate attractor values even in non-saturated recurrent cells, avoiding extreme states when warming-up. The procedure for computing \( VAA^* \) for all layers of an RNN is given in Algorithm 2.

We then use stochastic gradient descent to get the \( VAA^* \) of each layer as close as possible to \( k \in [0, 1] \). In practice, we use \( k = 0.95 \) as this proved, empirically, to maximize the number of attractors while avoiding too extreme states that could arise from the approximation of \( C \). The loss used is thus the following

\[
\mathcal{L}(\mathcal{U}, M, \theta) = \sum_{v_l \in VAA^*(\mathcal{U}, M, \varepsilon, \theta)} (v_l - 0.95)^2
\]

where \( VAA^*(\mathcal{U}, M, \varepsilon, \theta) \) represents the procedure for computing \( VAA^* \) with the corresponding parameters and data depicted in Algorithm 2 and where \( \sum \) represents an average over a set. Batches are sampled in the training set and to avoid over-fitting problems, \( M \) is sampled uniformly in \( \mathcal{U}(1, \ldots, M^*) \) at each gradient step. \( M^* \) is a variable initialized at 1 and increased by a constant \( \varepsilon \in \mathbb{N}_0 \) after each gradient step. This progressive increase, driven by the curriculum learning speed \( \varepsilon \), is required to smoothly reach multi-stability, avoiding gradient problems. Algorithm 3 details the whole warm-up procedure.

We show on Figure 2 that the warm-up procedure effectively increases the \( VAA^* \) of each layer in an RNN. Furthermore, we can also see on Figure 2 that as the warm-up procedure is carried out, the true \( VAA \) measure of the RNN also increases, even reaching 1 as the warm-up procedure ends.

**Double-layers.** Until now, we have only talked about attractors in RNNs. However, the literature (Sussillo and Barak, 2013) also points the importance of the transient dynamics of RNN for prediction. Indeed, it is easy to see why transient dynamics can be of importance when trying to tackle a regression task. If information is only stored in the form of attractors, then there can only be a fixed limit of states the network can remember, making it very hard to obtain precise predictions. We observed
Algorithm 2: Computing the set of $VAA^*$ for an RNN

**Data:** $\mathcal{U}$ a set of $n$ time-series $\{U_1, \ldots, U_n\}$ sampled in the training set.

**Parameters:** $M \in \mathbb{N}_0$ the number of time-steps used for state convergence.
$\epsilon \in \mathbb{R}_+^+$ tolerance when considering states similarity.
$\theta$ the architecture and parameters of the network.

**Result:** $V$ the set of $VAA^*$ of each layer for a random perturbation, computed on the given data.

```plaintext
/* Compute the initial states. */
X ← \{\}
foreach $U_i \in \mathcal{U}$ do
  $c \sim \mathbb{U}\{1, \ldots, T\}$ where $T$ is the length of $U_i$
  $x_i \leftarrow 0$
  for $t \leftarrow 1$ to $c$ do
    $x_i \leftarrow f(x_i, u_{c,i}; \theta)$ where $f(\cdot, \cdot; \theta)$ is the RNN's update rule.
  end
  $X \leftarrow X \cup \{x_i\}$
end

/* Use initial states to compute $VAA^*$ of each layer */
V ← \{\}
fforeach layer $l$ in $\theta$ do
  $u^l \sim \mathcal{N}(0, 1)$
  for $t \leftarrow 1$ to $M$ do
    $x^l_{i,t} \leftarrow f^l(x^l_{i,t}, u^l_{t}; \theta^l)$, $\forall x^l_{i,t} \in X$ where $f^l(\cdot, \cdot; \theta^l)$, $x^l$ and $\theta^l$ are respectively the update function, hidden state and parameters of layer $l$.
  end
  $\hat{C}_{i,j} \leftarrow 1 - \max(0, \frac{\|\tanh(x^l_{M,i}) - \tanh(x^l_{M,j})\| - \epsilon}{\|\tanh(x^l_{M,i}) - \tanh(x^l_{M,j})\|})$, $\forall i, j \in \{1, \ldots, n\}, i \neq j$
  $\hat{C}_{i,i} \leftarrow 1$, $\forall i \in \{1, \ldots, n\}$
  $v[i] \leftarrow \sum_{j=1}^{n} \hat{C}_{i,j}$ $\forall i \in \{1, \ldots, n\}$
  $VAA^* \leftarrow \frac{1}{n} \sum_{i=1}^{n} \frac{1}{v[i]}$
  $V \leftarrow V \cup VAA^*$
end
```

Algorithm 3: Warming-up an RNN

**Data:** $T$ a training set of time-series.

**Parameters:** $S \in \mathbb{N}_0$ the number of gradient steps.
$lr \in \mathbb{R}$ the learning rate.
$c \in \mathbb{N}_0$ constant driving the curriculum learning speed.
$M \in \mathbb{N}_0$ maximum convergence steps for $VAA$ computation.
$\theta$ parameters of the network.

**Result:** $\theta$ updates for multi-stability in different layers.

```plaintext
M^* \leftarrow 1
for $t \leftarrow 1$ to $S$ do
  $\mathcal{U} \leftarrow \mathbb{U}^n(T)$ where $\mathbb{U}^n(T)$ denotes a set of $n$ elements sampled uniformly in $T$.
  $L \leftarrow \mathcal{L}(\mathcal{U}, M^*, \theta)$
  $\theta \leftarrow \theta - lr \ast \frac{\delta L}{\delta \theta}$
  $M^* \leftarrow \min(M, M^* + c)$
end
```

Figure 2: Evolution of the $VAA^*$ for a two-layer RNN (left and middle) and of the $VAA$ of the network (right) during warm-up. This network was warmed up on the denoising dataset and results were averaged over three runs.
that when warming up neural networks, they tend to lose predictive accuracy, to the benefit of easier training on longer time-series. To alleviate this problem and obtain precise predictions while maintaining the benefits of warm-up, we propose using a double-layer architecture. We simply split each recurrent layer into two equal parts and only warm-up one of them, that is, we only compute $VAA^*$ on those parts of the network and solely update their variables when warming up. This enables the endowment of some part of each layer with multi-stability, while the rest remains mono-stable with richer transient dynamics. A double-layer structure is depicted on Figure 3.

4 Results

To demonstrate the impact of warming up RNNs, we tackle three benchmarks. These benchmarks were proposed in (Vecoven et al., 2020) to test the ability of RNN cells to learn long-term dependencies. The first is a one-dimensional toy problem, the second is a two-dimensional denoising problem, and the third is the permuted sequential MNIST problem. We train LSTMs (Hochreiter and Schmidhuber, 1997), GRUs (Cho et al., 2014) and MGUs (Zhou et al., 2016) without warm-up, with warm-up and with double-layer warmup (DLWU) on these benchmarks and show that their performance is greatly improved with warm-up, in a single or double-layer setting. Parameters for warm-up can vary depending on architectures and needs, however we found $lr = 1e^{-2}$, $c = 10$, $S = 100$ and $M = 200$ to be a good choice for an effective and fast warm-up on our benchmarks.

For the first benchmark, networks were made of one 128 neuron recurrent layer. For the other two problems, networks were made of two recurrent layers, each of 256 neurons. For the first two problems, training sets comprised 40000 samples and performances were evaluated on test sets generated with 50000 samples. For the permuted MNIST benchmarks, the standard train and test sets were used. All averages and standard deviations reported were computed over three different seeds and training was done with the ADAM optimizer and a learning rate of $1e^{-3}$. Concerning warm-up, we note that in some rare cases, if the $VAA^*$ is too close to 1 after warming up, RNNs become stuck and unable to learn. This is due to a bad fitting of the $VAA^*$ measure and thus internal states becoming too extreme. Although very rare, we solved this issue by restarting any run for which $VAA^*$ is too high for any layer after warm-up (we empirically chose 0.98 as a threshold in this paper).

**Copy first input benchmark** In this benchmark, the network is presented with a one-dimensional time-series of $T$ time-steps where $x_t \sim N(0, 1), \forall t \in T$. After receiving $x_T$, the network output value should approximate $x_0$, a task that is well suited for capturing their capacity to learn long temporal dependencies if $T$ is large. This benchmark allows for a simple proof of concept that warming up RNNs provides certain benefits for training. Indeed, one can see in Table 1 that warm-up greatly improves the performances of all RNNs as $T$ increases.

**Denoising benchmark** In the denoising benchmark, the network is presented with a two-dimensional time-series of $T$ time-steps. The first dimension is a noised input stream, where the value for each time-step is sampled from a normal distribution. Five time-steps of this stream should be remembered and output, one-by-one, by the network at steps $\{T - 4, \ldots, T\}$. These five
time-steps are sampled uniformly in $\mathbb{U}\{0, \ldots, T - N\}$ and are communicated to the network thanks to the second dimension of the input. $N$ is a hyper-parameter that allows for tuning how long the network should be able to retain information at a minimum. For a more precise description of the benchmark, see (Vecoven et al., 2020). In this benchmark, the amount of information the network must store is much greater as it needs to store five real values. This allows for a demonstration of the effectiveness of the double-layer architecture combined with warm-up. Indeed, Table 2 shows that warm-up is required for the networks to learn when $N$ increases. It is important to note that the standard deviation obtained for MGUs and LSTMs without double-layers is due to failed runs. That is, despite a successful warm-up, the network is not able to learn and loses its multi-stability properties. We believe this is due to the nature of the problem which requires precise transient dynamics for outputting the prediction. We see that adding a double-layer architecture solves this problem.

**Permutated sequential MNIST** In this benchmark, the network is presented with the MNIST images, where pixels are shown, one by one, as a time-series. It differs from the regular sequential MNIST in that pixels are shuffled, with the result that they are not shown in top-left to bottom-right order. This benchmark is known to be a more complex challenge than the regular one. Table 3 shows that warm-up with a double-layer architecture provides equivalent performances than without warming up. The slight decrease in GRU performance can be explained by the lower number of parameters in the double-layer architecture as compared layers connected fully recurrently. This benchmark shows the importance of a double-layer structure when warming up RNNs.

### Table 1: Mean square error ($\pm$ standard deviation) of different architectures and different warm-up strategies on the test set for the copy input benchmark. Results are shown after 50 epochs and for different values of $T$.

<table>
<thead>
<tr>
<th>$T$</th>
<th>Warm-up</th>
<th>MGU</th>
<th>LSTM</th>
<th>GRU</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>None</td>
<td>0.831 ± 0.324</td>
<td>0.634 ± 0.427</td>
<td>0.997 ± 0.005</td>
</tr>
<tr>
<td></td>
<td>warmed-up</td>
<td>0.000 ± 0.000</td>
<td>0.000 ± 0.000</td>
<td>0.000 ± 0.000</td>
</tr>
<tr>
<td></td>
<td>DLWU</td>
<td>0.000 ± 0.000</td>
<td>0.000 ± 0.000</td>
<td>0.000 ± 0.000</td>
</tr>
<tr>
<td>300</td>
<td>None</td>
<td>0.98 ± 0.002</td>
<td>0.95 ± 0.019</td>
<td>1.003 ± 0.002</td>
</tr>
<tr>
<td></td>
<td>warmed-up</td>
<td>0.000 ± 0.000</td>
<td>0.000 ± 0.000</td>
<td>0.000 ± 0.000</td>
</tr>
<tr>
<td></td>
<td>DLWU</td>
<td>0.000 ± 0.000</td>
<td>0.000 ± 0.000</td>
<td>0.000 ± 0.000</td>
</tr>
<tr>
<td>600</td>
<td>None</td>
<td>1.017 ± 0.004</td>
<td>0.977 ± 0.008</td>
<td>1.017 ± 0.003</td>
</tr>
<tr>
<td></td>
<td>warmed-up</td>
<td>0.000 ± 0.000</td>
<td>0.000 ± 0.000</td>
<td>0.000 ± 0.000</td>
</tr>
<tr>
<td></td>
<td>DLWU</td>
<td>0.000 ± 0.000</td>
<td>0.000 ± 0.000</td>
<td>0.000 ± 0.000</td>
</tr>
</tbody>
</table>

### Table 2: Mean square error ($\pm$ standard deviation) of different architectures on the denoising benchmark’s test set. Results are shown with and without constraint on the location of relevant inputs and after 50 epochs. Relevant inputs cannot appear in the $N$ last time-steps. In this experiment, results were obtained with $T = 200$.

<table>
<thead>
<tr>
<th>$N$</th>
<th>Warm-up</th>
<th>MGU</th>
<th>LSTM</th>
<th>GRU</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>None</td>
<td>0.005 ± 0.008</td>
<td>1.001 ± 0.003</td>
<td>0.000 ± 0.000</td>
</tr>
<tr>
<td></td>
<td>warmed-up</td>
<td>0.002 ± 0.006</td>
<td>0.032 ± 0.011</td>
<td>0.000 ± 0.000</td>
</tr>
<tr>
<td></td>
<td>DLWU</td>
<td>0.002 ± 0.001</td>
<td>0.025 ± 0.007</td>
<td>0.000 ± 0.000</td>
</tr>
<tr>
<td>100</td>
<td>None</td>
<td>1.004 ± 0.003</td>
<td>0.996 ± 0.005</td>
<td>0.995 ± 0.003</td>
</tr>
<tr>
<td></td>
<td>warmed-up</td>
<td>0.32 ± 0.641</td>
<td>0.338 ± 0.561</td>
<td>0.001 ± 0.002</td>
</tr>
<tr>
<td></td>
<td>DLWU</td>
<td>0.024 ± 0.023</td>
<td>0.013 ± 0.125</td>
<td>0.000 ± 0.000</td>
</tr>
</tbody>
</table>

### Table 3: Overall accuracy ($\pm$ standard deviation) on the permuted sequential MNIST benchmark test set after 70 epochs for different warm-up methods and for different cell types.

<table>
<thead>
<tr>
<th>Warmup</th>
<th>MGU</th>
<th>LSTM</th>
<th>GRU</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>0.896 ± 0.004</td>
<td>0.907 ± 0.002</td>
<td>0.925 ± 0.004</td>
</tr>
<tr>
<td>warmed-up</td>
<td>0.897 ± 0.001</td>
<td>0.402 ± 0.012</td>
<td>0.102 ± 0.061</td>
</tr>
<tr>
<td>DLWU</td>
<td>0.901 ± 0.003</td>
<td>0.909 ± 0.005</td>
<td>0.914 ± 0.014</td>
</tr>
</tbody>
</table>
5 Conclusion

In this paper, we propose warming-up recurrent neural networks to improve their ability to learn long time-dependencies. The procedure is motivated by recent work that showed the importance of fixed points and attractors for the prediction process of trained RNNs. We introduced a lightweight measure called VAA, that can be optimized to endow RNNs with multi-stable dynamics. Warm-up can be used with any type of recurrent cell and we show that it vastly improves their performance on long-term memory benchmarks when combined with a double-layer architecture. As this procedure is general and easy to implement, we believe it can easily be further tested on multiple benchmarks.

As future work, an area of application that is promising for such an approach is sparse reinforcement learning. In fact, warming up RNNs would allow them to remember information for much longer, and thus be more robust to complex exploration in such a setting.

Furthermore, we note that the double-layer architecture might be worth exploring with different types of cells. We show here that there are benefits of using different types of initialization for the same type of cell. This might hint at the possibility of having similar benefits when using different types of cells, each with different dynamical properties, in RNNs.

Finally, in this paper, we aimed to maximize the number of attractors through warming up before training. We noticed, however, that in some rare cases, networks lose multi-stability properties when training. Using VAA as a regularization loss to avoid this could be interesting. Likewise, when warming-up, one could choose to aim for a number of attractors that is optimized for a given benchmark. This provides further room for improvement of algorithm performance.

Acknowledgments and Disclosure of Funding

Nicolas Vecoven gratefully acknowledges the financial support of the Belgian FRIA.

References


